Prediction of non-functional properties of (Component) COTS-based Systems: A Model for Predicting Software Reliability

Adel Taweel

Department of Computer Science,
University of Manchester, Manchester, M13 9PL, UK
e-mail: A.Taweel@cs.man.ac.uk

Abstract
On the outset, components or COTS-based systems offers several advantages, including the promise of shorting the development life cycle, reducing costs of software development and faster utilisation of recent technical improvements in software industry in terms of capability, reliability, compatibility, performance and so forth. Many (initially!) believed that developing COTS-based systems would be similar in many ways to LEGO-like building and would easily deliver these promises. However, this is rarely realised in practice. An important issue of COTS-based systems is that they are likely to be (or perhaps already being) used in domains where human life and/or economic loss are possible and the need for a highly reliable system is a must. However, would it be possible to build reliable COTS-based systems that meet such domains’ requirements? Does it necessarily mean that composing a system from highly reliable COTS components produce a highly reliable system? Is it possible to predict quality attributes of a COTS-based system from its components before building it? In this paper we highlight the complexity of this issue, focusing on reliability, and attempt to answer these questions. We outline various approaches that attempted to address this issue, and propose a possible way forward to predicting the reliability of components or COTS-based system from its individual components.

1. Introduction and background

Traditionally, software systems are constructed from individual parts, which are then integrated to produce a system that meets certain functional and non-functional specifications. The produced system as a whole is, then, tested to check whether these specifications are met. Systems in this case are normally viewed as a whole, and thus a whole system, rather than its individual parts, is tagged by these specifications. Although changes may need to be done on certain parts of the system to improve a non-functional property, for example, the whole system will have to be re-tested to check whether or not that improvement has been achieved.

In COTS or component-based architectures, however, systems are constructed from individual components. These individual components are combined together, normally based on a well-defined architecture or framework, to produce a system that, as well, meets certain functional and non-functional specifications or properties. These individual components, on the other hand, have their own functional and non-functional properties. Which leads to the question: whether the properties of individual components can be used or extended to determine the properties of the composed system?
One way to determine the properties of the composed system is to re-test this composed system as whole. However, in component-based systems, this method is not always feasible and definitely not preferable for several reasons. Firstly, with components are expected to be bought (or obtained) individually, possibly from different vendors, to compose a system, it is unlikely that developers (or integrators) would commit themselves to buying a component that they intend to use in a system and then it turns out to not achieve what it was originally bought for. The terms developers and integrators are used interchangeably throughout the paper. Secondly, even if it was possible to acquire a component without buying it, it is less likely that integrators would be happy to lose their effort because of a component was wrongly selected. Thirdly, it is not economical to integrators as they are expected to meet deadlines and budget.

In reality, integrators are unlikely to choose a component to test or evaluate for their system unless it stands a fair chance of success. Determining how successful would be a selection of a component is a complex task, and if there exist a mechanism that indicates in one way or another that chance of success, it would greatly reduce the complexity of the selection process and thus the development of the system. This mechanism could be envisaged from enabling developers (or integrators) to determine or predict the properties of the resultant composed system from the properties of selected individual components. The potential and importance of prediction theories for COTS or component-based systems has been recognised in the software engineering research community and in fact, special teams have shown special interest in this field [Bachmann00, Crnkovic01, Wallnau03].

In general, components will possibly be selected based on their functionality to achieve in a way, one or more of the pre-defined specifications of the goal composed system. Several researchers suggested different methods for composing (or facilitating the composition of) a component-based system through the use of a formal composition language [Lumpop00, Lump97] or a composition architecture or framework [Plakosh99]. These methods mainly define a mechanism for gluing components based on their functional abilities and through using standard interface. Thus, because component-based systems are built from individual component to satisfy certain functional specifications or functional abilities, which are basically derived from the functional properties of individual components, there is no urgent need to predict these properties. In addition, functional properties of individual components can be extended to composed systems, simply because they are represented by actions and not values! However, the challenging task will to be able to predict non-functional properties of COTS or component-based system. Thus, the next questions that arise are how to determine or predict these non-functional properties? Whether or not the existing theories are sufficient in their current state? And what are the obstacles to predicting non-functional properties? These questions are explored below.
2. Issues and complexities

There are several obstacles and issues related to component-based systems that should be addressed by any approach that attempts to predict final system properties. These are discussed below.

Non-functional properties representation: as mentioned above one of the main difficulties that will be faced with when addressing these questions will be determining how these non-functional properties are represented, before even starting addressing various techniques that can be used to compose these properties or determine the resultant system non-functional properties. Bachmann et al highlight this problem through asserting the importance of certifying components [Bachmann00]. Certification does not only provide tangible representation of non-functional properties, but also provides minimal guarantees in a way about how components will behave (functionality) [J. Voas in Crnkovic01]. Therefore, it is essential to study different certification techniques and various representations of values of these non-functional properties before envisaging possible techniques that can be applied on these representations. Although Bachmann et al expressed the need for certification, they also asserted that until there exists a strong theory that utilises certification results to predict composed system properties, industry would not be enthusiastic to certify their components. However, it is not necessary to certify all non-functional properties of components, as some properties do not contribute to the properties of the composed system. For example, quality of documentation of individual components does not necessarily contribute to the quality of documentation of the composed system.

Composition architecture/framework: In component-based systems, components are integrated together to form a system. Since these components are expected to be self-contained and have the capability of operating standalone, they must have or provide an interface capability through which they can be invoked or accessed. Integrating these components together thus may follow a particular architecture; this is also referred to as glue logic. This architecture can be very intelligent and complicated and acts as an operating system that supports multiple interfaces and have the capability of defining various glue methods between components and, at the same time, defines constraints on components deployment. It can be seen as an operating system or a motherboard that supports plug and play, in which components can be plugged in to form a system that its operations are decided by a microprocessor or main component. Figure 1 depicts such architecture. On the other hand, this glue architecture can be as simple as a traditional “main program” that includes a set of instructions that defines the sequence of how components are invoked (see Figure 2). However, in either case, this composition or glue architecture adds further constraints and additional overheads that are bound to affect the whole system properties. For example, reliability of the final system will, as well, be affected by how components interact, which is determined by the glue architecture. Several questions arise, such as, which architecture that adds less overheads on the final system properties (especially non-
functional properties), and how these overheads can be calculated or evaluated etc. Therefore any theory that attempts to predict final system property should as well address composition architecture properties and their effect on the final system.

Several researchers highlighted the need for addressing the properties of the employed composition architecture or framework [Bachmann00, Lycett99, Klein99]. The properties of the underlying composition architecture or framework, such as interconnection and communication mechanism, are bound to affect and/or determine properties of the composed system. Bachmann et al suggest that it could be as well necessary to study the composition framework properties, types of composition frameworks and their attributes [Bachmann00]. In addition, looking at improving the composition framework non-functional properties may as well contributes more to the composed system than those of individual components [Lycett99].

**System structure**: Systems are built to solve a particular problem or to meet a given set of requirements. Components are selected and then integrated together in a particular structure to form the target system that meets these requirements. The same components could be re-arranged or re-structured to meet the same requirements resulting in many possible system structures of these components that offer the same solution. Similarly, the question that arises is whether re-arrangements of components with a system (or system structures) affect final system properties? And if so, how these effects can be represented? It is quite logical to also expect that a particular system structure may positively affect a particular non-functional property and negatively affect another, where a compromise is then sought.

**Operational profile** defines the environment that components operate in and the input data that these components use to carry out it operations. In one case, systems could be built from components that are designed to work in different environments, forming for example distributed systems on different operating systems and/or hardware systems. In another case, systems could be built from components that can work under the same environment. In either case, components are affected by their operational profile, however, how they are
affected and how these contribute or affect the final system properties are important aspects and should be addressed in a more quantifiable or measurable way by the predication theory.

**Level of dependency between components:** although in COTS or component-based systems components are, generally, expected to be standalone and self-contained, components can depend on each other within a system to complete their operations. A component could use functions or services provided by other components to provide a new function or service. This structure forms architecture of subsystems that depend on other subsystems, which eventually as a whole constitute the final system. However, if a component within a subsystem fails, that subsystem may fail as a whole to provide its services especially if its components depend on each other to provide that service. How often that subsystem fails to provide a service depends on how dependent is that system on the failed component. Thus the level of dependency between components eventually affects the final system being able to provide its services. Predicting the reliability, for example, of a final system would depend on how well this dependency is captured by the predicting theory.

**Level of knowledge required to develop a “sufficient” prediction theory:** the different models discussed below require different levels of knowledge about a component or components of a system to provide a reasonable predication. However, for any theory to become practical and widely adopted, it should be applicable to various types of components with the diversity of levels of knowledge available about them. Off-the-shelf components, for instance, exist in the market place in different forms, with most as closed-source and many others as open-source. Some have reasonable documentation, others do not. But, in general, the knowledge available about these components are very limited indeed. On the other hand, components that are under development could consider this point and hence could be packaged with sufficient level of knowledge to meet most predication models. In fact, some of these models demand components to be designed according to certain architectural styles to be applicable [Klein99]. While predication models might demand high levels of (internal) knowledge about components to be “sufficient”, they should also be developed with reasonable assumptions that consider the state of the existing reality of the market.

**Frequency of component usage in a system:** The structure of components within a system determines how often components are used to provide a particular service or provide all services. Components that have higher frequency of usage thus determine how often a system can provide its services. It is logical therefore to expect that properties of these components should have higher influence on the final system properties. Determining these components as well as their corresponding amount of influence on the final systems may prove very advantageous. For example, if in a system, 80% of its operations depend on a particular component, then the reliability of this component is expected to contribute more to the final system reliability than a component that only used in only 5% of its operations.
A similar example can be made on the performance property. Of course cost and time effectiveness (for maintainability for instance) can as well be drawn from such measures.

**Methods used to calculate/measure a non-functional property of components:** non-functional properties (often referred to as *ilities*) are difficult concepts in nature; finding the best ways of representing and measuring these properties have always been a challenge to the software engineering community. This has resulted in several methods (or metrics) being proposed for measuring (and/or representing) each of these properties of a system. For example, as mentioned below, there are several approaches to measure reliability of a component or a system. While predicting reliability of a composed system from its components’ reliability that have used the same method of measurement may prove a challenge, determining reliability of the composed of components’ reliability which were measured used different methods of measurement may yet prove a harder challenge. However, since no one method has been selected as a standard and unless one is selected, in practice that may very well happen. Of course, one solution would be that reliability of components could be provided in different measurements (using methods). Another solution could be that, the same method used to predict the final system properties could be extended to predict properties of individual components, considering them built of sub-components.

### 3. General Approaches

There are two general approaches to predicting non-functional properties. These can be classified as *black-box* (or *macroscopic*) approaches and *white-box* (or *microscopic*) approaches. In *black-box* approaches, the (system-level) properties of a component-based system are predicted from (high level) properties of its individual components, dealing with each component as a “black-box”. These approaches examine the (high level) properties of these individual components and study their impact on system-level properties attempting to predict an equivalent property of the system while avoiding the details of how these properties are obtained. Model checking is an example of *black-box* approaches. Model checking is normally used for automatic model verification and validation against requirement specifications. It employs LTL (Linear temporal logic) and CTL (Computational Temporal logic) to check, what normally referred to as, *correctness of model*. However, it can be extended to also deduce system properties from their own components’ properties by using additional interface processes for establishing correctness of (system) properties [Clark89]. Clark et al proposes a theory that employs CTL (Computation Tree Logic) logic to deduce system properties out of the properties of its components. The authors refer to component properties as *local properties* and systems properties as *global properties*. However, to enable preserving these local properties at a global level he suggests using additional *interface processes* to model the environment for components. Composing these interface processes with components of a system should then
create an integrated system that enable preserving properties of individual components at a
global level.

In the *white-box* approaches, however, not only the high-level properties of individual
components are taken into consideration but also details of these properties are studied to
examine their impact on system properties. Some of these approaches specify how
components should be designed; others demand specific details that should be collected
during components development cycle to enable better prediction of system-level properties.
*Compositional reasoning* theory is an example of this approach. *Compositional reasoning*
theory employs mainly algebra and logic to predict the end-system properties. This theory is
based on the belief that end-system properties are most often attributable to interacting
components within the system and thus the properties of these components can be
“composed” to predict properties of the end-system. ABAS is an example that uses
compositional reasoning to deduce system performance out of its components’ performance
[Klein99]. It is based on the assumption that a specific architecture or design is employed
for the components used. It defines a set of architecture styles and design guidelines that
components should use. Based on these guidelines, it sub-classifies system attributes and
then *reasons* the impact of changes in architecture design on these attributes. In other words,
it tries to formalise architecture design as an engineering method that should be followed for
developing components.

4. Towards a Solution

A solution for predicting non-functional properties of composed system from the properties
of their individual components could be envisaged from prediction theories, such as
*compositional reasoning* [Bachmann00]. However, existing *compositional reasoning*
theories, such as ABAS, imply that components should have been developed according to
pre-defined architecture styles and/or design guidelines, which would make them
inapplicable onto off-the-shelf components (COTS). In fact, most of COTS exist as binary
files, with little or no knowledge available on their internal structure or design. Although a
significant number of COTS exist as an open-source, few will be tempted to dig into them
and look closely at how it was designed. The Majority of users will be interested in using
off-the-shelf components rather than completely comprehending their designs or re-
developing them. On the other hand, recent research has lead to the creation of what referred
to *component-based software engineering* (CBSE) that attempt, in one way or another, to
“formulise” the engineering of components so that it eventually enables the development of
prediction-enabled component-based systems [Bachmann00, Crnkovic01]. Several models
have been proposed that demand certain (internal) knowledge about components and put
constraints on their designs [Hamlet01, Klein99]. Nevertheless, a *successful* COTS or
component-based predication theory should take in consideration the amount of effort
needed to be spent on assessing potential components before selection.
Predicting system-level non-functional properties process can be envisaged as one depicted in the following figures. Figure 3 views this process as a compositional process where the same properties of each component are composed or collapsed to produce the equivalent property for the whole system. This is similar to the compositional reasoning and model checking views. For example, in ABAS, the performance property of individual components are composed to produce the equivalent performance property for the whole system, though it adds some constraints on how the system should be designed and on which architecture should be used.

Figure 4 views this process as a network aggregation process where an output system-level property is seen as a contribution of not just the equivalent component-level property but also other component-level properties. This forms a network-like skeleton that at one end has the component-level properties and on the other end system-level properties. For example, to find out the reliability of a system, we may need not just to look at the reliability of individual components only, but also at their fault tolerance and recoverability attributes. This also may view the system as a whole aggregated of sub-modules rather than isolated components. Network theories can probably be recalled that address similar problems: Petri nets. Petri nets, although used to aggregate processes, can probably be seen as a potential method to aggregate non-functional properties especially when the process is depicted as a network.

The process can as well be viewed as a conversion process (see Figure 5), where a relevant or equivalent set of component-level properties are aggregated or converted to an equivalent system-level property. For example, the reliability of a system can be seen as made up of reliabilities of individual components.
In another view, the reliability of a system can be seen as related to reliabilities of its system-level functions, where the reliability of each system-level function is an aggregation of reliabilities of contributing component-level functions. This yields chains-like structure for each system level property for each function. Theories related to stochastic Markovian chains can probably be of relevance. Such theories attempt to aggregate each Markovian chain within a system into a single value.

Since, however, measuring all non-functional properties of systems or components does not follow a particular method, metric or a standard, addressing all these properties using the same theory may not prove a viable approach. In fact, measuring non-functional properties of components or systems (as whole) still poses a challenge to the computing community. Most of the existing methods or theories from metrics or formal methods disciplines offer only prediction and not accurate measurement. Therefore, addressing each non-functional property individually not only enables extending existing predictive methods or theories but also enables using theories from other disciplines and simplifies the complexity of the problem. For example, as discussed above, theories from stochastic process algebra, aggregation of Markovian chains (operational research) and Petri nets, address similar problems and might be extended to offer a better solution to this problem.

Therefore, the rest of the paper focuses on theories that attempt to predict reliability of COTS or component-based system.

5. Predication of Reliability of Component-based System

Reliability of a software system is defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time [IEEE90]. Reliability of a software system is also defined as its probability of failure free operation in a given environment [Krishnamurthy97]. Approaches for measuring reliability can be similarly classified into two main types: black-box and white-box. Black-box approaches attempt to measure or estimate reliability of a software system while dealing with it as a “back-box” and not needing any internal knowledge of the software system. However white-box or microscopic approaches uses the data gathered during the development stage of a software system and attempts to estimate its reliability. Nevertheless both approaches have been criticised as fraught with risk and not entirely satisfactory for use in practical software development environments [Horgan95, Gokhale97]. However, estimating or knowing software reliability measures can be very useful. For example, knowing software reliability measures allows software users and developers alike to be selective about the software they want to buy or use. In fact, knowing software reliability measures yet becomes more important for component-based systems. Such measures help developers or integrators not only to be selective about the components they use but also help them to guess or predict such measures for their final system. For example, if
components with very low reliability have been used, what chance would the final system have high reliability?

On the other hand, it would be of great advantage if there exists a theory that can utilise components reliability measures and predict the final system reliability. Several approaches that have been proposed to measure software reliability can be considered of relevance to component-based systems [Dolbec95, Woit98, Krishnamurthy97, Hamlet96, Hamlet01]. Dolbec’s and Kirshnamurthy’s approaches can be considered black-box approaches, while Hamlet’s approach can be seen as microscopic approach. The first three approaches use the reliability measures of individual components to predict the reliability of the final system; the latter, however, demands internal knowledge of components through the collection of data during the development stage and utilises this data to predict the reliability of the final system. While this model might be more accurate (as claimed by Hamlet), it is not practical and in most cases inapplicable especially for COTS and component-based system and unlikely to be favoured by industry. On the other hand, Krishnamurthy’s and Doblec’s models are black-box approaches that attempt to predict systems reliability out of their components or modules. Although such models might produce less accurate results but they are applicable to a wider range of (especially off-the-shelf) components and are more likely to be favoured or used by practitioners. However, due to the accuracy of such models in their current state, they are unlikely to be widely adopted.

5.1. Our Approach

Krishnamurthy’s and Doblec’s models deal with each module or component as a “black-box” and use the reliability associated with a component as whole to predict the reliability of the system. However, in COTS or component-based systems, since components are expected to be self-contained and can, in many cases, operate standalone to provide certain functions or services and since these services will not necessarily all be used (as systems may utilise only part of these services), it is more appropriate and accurate to associate properties to functions or services of a component rather than to a whole component. In this context, the terms function and service are used interchangeable in the remainder of the paper unless otherwise mentioned. In addition, if components are expected to be fully self-contained then level of dependency between components may become negligible which then reduces the complexity of predicting the final system properties. Consequently, the values of properties will be associated with the functions or services provided by the final system. Such model is not completely a black-box approach, since it considers reliability of individual functions provided by each component opposed to the component as a whole. In this case, the reliability of a service provided by the system can be derived from reliabilities of the corresponding component functions used as part of that service. Thus a COTS or component-based system can be viewed as made up of services and these services are made
of execution paths that run across the components of the system. We refer to this approach as Service-Based Reliability Estimation (SBARE) model.

In addition, since functions can be considered as (behavioural) subparts of components, theories used in Krishnamurthy’s and Doblec’s models can be modified or extended to apply to this model. Although both models follow the same general concept, their approach to deriving the theory is different. Therefore, based on these two models, two corresponding models can result. Further, theoretical and empirical research can, then, be further applied to evaluate when and which model provides more accurate results. The following derivation assumes that the composition architecture is as depicted in Figure 5, where a “main program” that defines the sequence of how components are invoked and the behaviour of the system. The behaviour of the system is determined by the functions or services it provides. SBARE attempts to tag each function with its own reliability measure. In this case, the main program is considered as a component and as part of execution paths of each system-level service. This model can, as well, be applied to the composition architecture depicted in Figure 4. Although, in this case, there are additional components included by the architecture, these components should be considered as part of the system. Some or all of these components might be part of every execution path of every system-level function. However, since these additional components may impose a known overhead from one system to another, its properties and their effect on any system is more likely to be measured.

### 5.1.1. Extending Doblec’s model

The following derivation is based on Doblec’s model. Doblec’s model is based on structure based software reliability model proposed by Shooman [Shooman91]. In Shooman’s model, the number of failures in a software system can be found by the following equation.

$$n_f = N u_i q_i + N u_2 q_2 + \ldots + N u_l q_l$$

Where $N$ is the number of software system tests, $u_i$ is the usage ratio of the execution path $i$, $q_i$ is the probability of failure of path $i$ and $l$ is the number of execution paths. This equation (0.1) suits very well the SBARE model view, as it views the system as made up of execution paths. Obviously the probability of failure (unreliability) of the software system can be found as

$$Q_s = \frac{n_f}{N} = \sum_{i=1}^{l} f_i q_i$$

(0.2)

and the reliability of the software system can be found as

$$R_s = 1 - Q_s$$

(0.3)
Equation (0.2) represents the unreliability of a software system as equal to the sum of the probability of failure of every execution path within this system weighted by its corresponding usage ratio.

To extend this theory on a similar basis to Dolbec’s model, however for service-based reliability estimation consider the following.

\( C_{iF,jK} \) is event that function \( F_j \) in component \( C_i \) of path \( K \) executes successfully.

\( p(C_{iF,jK}) \) is the probability of function \( F_j \) in component \( C_i \) of path \( K \) does not fail.

\( q(C_{iF,jK}) \) is the probability of function \( F_j \) in component \( C_i \) of path \( K \) does fail.

\( E_{KF} \) is event that execution path \( K \) within function \( F \) is successfully executed.

\( p(E_{KF}) \) is the probability of execution path \( K \) within function \( F \) does not fail.

\( q(E_{KF}) \) is the probability of execution path \( K \) within function \( F \) does fail.

Since each system-level function can be made up of one or more execution path, the reliability of each execution path can be found by multiplying the reliability of component-level functions used within that path. However to avoid complications in the calculation, it is assumed that these execution paths are independent and failure of one function in a component does not cause a failure in another function with the same component.

Thus probability of execution path \( K \) within function \( F \) does not fail can be found as

\[
p(E_{KF}) = \prod_{i=1}^{n_K} \prod_{j=1}^{m_K} p(C_{iF,jK})
\]  
(0.4)

Where \( n_K \) is the number of components and \( m_K \) is the number of functions within path \( K \) (\( n_K \) and \( m_K \) have the same value, i.e. \( n_K = m_K \)).

Equation (0.4) can be expressed as

\[
p(E_{KF}) = \prod_{i=1}^{n_K} \prod_{j=1}^{m_K} 1 - q(C_{iF,jK})
\]  
(0.5)

Consequently, the probability of failure of execution path \( K \) within function \( F \) can be found as

\[
q(E_{KF}) = \prod_{i=1}^{n_K} \prod_{j=1}^{m_K} 1 - p(C_{iF,jK}) = \prod_{i=1}^{n_K} \prod_{j=1}^{m_K} q(C_{iF,jK})
\]  
(0.6)

However if reliabilities of component-level functions within execution path \( K \) are high, then equation (0.5) can be approximated (as shown in [Misra92]) to the following equation
\begin{equation}
\begin{align*}
p(E_{KF}) & \approx 1 - \sum_{i=1, j=1}^{n_k m_k} q(C_{ifjK}) \\
\text{(0.7)}
\end{align*}
\end{equation}

Consequently, the probability of failure can be written as

\begin{equation}
\begin{align*}
q(E_{KF}) & \approx \sum_{i=1, j=1}^{n_k m_k} q(C_{ifjK}) \\
\text{(0.8)}
\end{align*}
\end{equation}

Using this equation (0.8) in equation (0.1) the number of failures of a given system-level Function \( F_{w,s} \) can be found as

\begin{equation}
\begin{align*}
n_f F_{w,s} & \approx Nu_1 (\sum_{i=1, j=1}^{n_1 m_1} q(C_{ifj1})) + Nu_2 (\sum_{i=1, j=1}^{n_2 m_2} q(C_{ifj2})) \\
& + ... + Nu_K (\sum_{i=1, j=1}^{n_K m_K} q(C_{ifjK})) + ... + Nu_l (\sum_{i=1, j=1}^{n_l m_l} q(C_{ifjl})) \\
\text{(0.9)}
\end{align*}
\end{equation}

Where \( l \) is the number of execution paths within Function \( F_{w,s} \) and \( w \) is the function number within the system \( s \).

Therefore the unreliability of \( F_{w,s} \) can be found as

\begin{equation}
\begin{align*}
Q_{F_{w,s}} & \approx \frac{n_f F_{w,s}}{N} = u_1 (\sum_{i=1, j=1}^{n_1 m_1} q(C_{ifj1})) + u_2 (\sum_{i=1, j=1}^{n_2 m_2} q(C_{ifj2})) \\
& + ... + u_K (\sum_{i=1, j=1}^{n_K m_K} q(C_{ifjK})) + ... + u_l (\sum_{i=1, j=1}^{n_l m_l} q(C_{ifjl})) \\
\text{(0.10)}
\end{align*}
\end{equation}

In general, same functions could be executed or utilised by different paths within the same system-level function. Therefore, \( C_{1fj1} \) and \( C_{3fj2} \) can be the same function. Thus some functions will have more usage rates than others. Consequently if we combine the same components by adding their individual usage ratios, equation (0.10) can be re-written as

\begin{equation}
Q_{F_{w,s}} \approx U_1 Q_1 + U_2 Q_2 + ... + U_y Q_y \\
\text{(0.11)}
\end{equation}

Where \( y \) is the number of component-level functions used in the execution of the system-level function \( F_{w,s} \). \( U_y \) is the usage ratio of component-level function \( y \) in \( N \) tests. \( Q_y \) is the probability of failure of component-level function \( y \). Equation (0.11) can be re-written as

\begin{equation}
Q_{F_{w,s}} \approx \sum_{i=1}^{y} U_i Q_i \\
\text{(0.12)}
\end{equation}
Consequently reliability of function $F_{w,s}$ can be found as

$$R_{F_{w,s}} = 1 - Q_{F_{w,s}} = 1 - \sum_{i=1}^{V} U_i Q_i$$

(0.13)

This equation implies that reliability of a system-level function can be found from the sum of un-reliabilities of used component-level functions weighted by their corresponding usage ratios. These usage ratios also can be used as a useful indication of component-level functions that has higher impact on the corresponding system-level function. In Dolbec’s model, he defines the usage ratio as equal to the component execution time divided by the total system execution time. However, on a function level, we can redefine this usage ratio as the number of times a given (component-level) function is used in the execution of a system-level function divided by the total number of times all other (component-level) functions are used, i.e

$$U_{F_j} = \frac{N_{C_{F_j}}}{N_{F_{w,s}}}$$

(0.14)

Where $N_{C_{F_j}}$ is the number of times a component-level function $F_j$ used in the execution of the system-level function $F_{w,s}$. $N_{F_{w,s}}$ is the number of times all other (component-level) functions are used in the execution of the system-level function $F_{w,s}$.

This is reasonable assumption, since we are interested only to know whether a function completes its function or not and not necessarily how long it takes to complete it. In addition, in practice, it will be hard to find out individual components’ execution time not because it a short period to measure but also this time is dependent on which functions are used and their operational profile.

Of course it is also possible that some component-level functions are used in more than one execution paths across different system-level functions. Although we have assumed execution paths are independent, component-level functions that have higher usage ratios across system-level functions can provide a useful indication of impact of such functions on how many system-level functions and thus indicate their importance to the whole system.

### 5.1.2. Extending Krishnamurthy’s model

The above derivation is base on Shoorman’s model and further used the same approach followed in Dolbec’s model. A SBARE model can also be derived in a similar way as Krishnamurthy’s model. As an addition to Krishnamurthy’s model, an approach is suggested to represent the dependence between components, although, as reported case studies indicate, it is not always accurate. This is mainly due to the fact that system structure is not taken into account and the type of functionality of components is not considered as related to
the degree of dependency between components. However, SBARE model can reuse the experimental set up indicated in Krishnamurthy’s model to compare both derivations and provide a suitable evaluation for both.

The above derivation is based on the assumption that component functions have high reliability, thus for component functions with low reliability equation 0.7 to 0.14 are not applicable. However, equation 0.6 is still applicable. In fact, Krishnamurthy’s model builds on a similar equation. However, he assumes that a number of tests that should be carried out on system vary and sufficient number of test should be carried out that should eventually converge to a near true reliability.

Repeating equation 0.4, the probability of execution path $K$ within function $F$ does not fail can be found as

$$p(E_{KF}) = \prod_{i=1, j=1}^{n_K - m_K} p(C_{iF_jK})$$

If this execution path is tested by a given test case $t$, then using the above equation the reliability of execution path $k$ can found as

$$R'_{EKF} = \prod_{i=1, j=1}^{n_K - m_K} R'_{C_iF_jK}$$

Therefore, based on Krishnamurthy’s model, if all execution paths within the system-level function $F$ are tested with respect to a test set $T$, its reliability can be found as

$$R_{F_w s} = \frac{\sum_{\forall t \in T, k=1}^{K} R'_{E_kF}}{|T|}$$

Where $K$ is the number of execution paths within system-level $F_{W_s}$. The error $\varepsilon$ in this estimation can be found as

$$\varepsilon = \frac{R_{F_w s} - R_{TF_w s}}{R_{TF_w s}}$$

Where $R_{TF_w s}$ is the true reliability for function $F_{W_s}$. If $R_{TF_w s}$ is achieved through $N_{F_w s}$, where $N_{F_w s}$ is the number of tests or executions of function $F_{W_s}$ to arrive at $R_{F_w s}$ and $N_{TF_w s}$ is the number of tests or executions of $F_{W_s}$ to arrive at $R_{TF_w s}$, then the $\delta$-efficiency of this estimate for $\varepsilon \leq \delta$ can be found as

$$\eta_\delta = 1 - \frac{N_{F_w s}}{N_{TF_w s}}$$
\( \eta_\delta \) can be used as a measure of how fast \( R_{F_W} \) can be obtained compared to \( R_{FF_W} \).

In Krishnamurthy’s model dependency between components is handled by their degree of independence of components. This degree of independence of components determines the number of times its reliability is used in equation (0.15) compared to the number of its occurrence in an execution path or paths of the system-level function \( F_{W_S} \).

5.1.3. Validation and Evaluation

Another part of this research is the validation and evaluation of the above models, which have not yet been completed. In addition to comparing both models (section 5.1.2 and 5.1.3), the planned evaluation is to include both specifically designed components and off-the-shelf components, so that the applicability, accuracy and efficiency of the models are assessed.

However, since the above models are based on Dolbec’s and Krishnamurthy’s models, which both have been validated and evaluated, we expect to obtain similar results in terms of efficiency and accuracy. The difference, on the other hand, especially in the granularity of the above SBARE models being at the function-level rather than at the component-level, is anticipated to generally contribute, improve and generate results with higher efficiency and accuracy and with a wider spectrum of values across the horizontal and vertical dimensions of the composed system.

6. Conclusion

Predicting non-functional properties of COTS or component-based systems from properties of their components faces many challenges with a complicated mix of obstacles and related issues, yet, if successful, promises significant potential benefits to users, developers and industry. These promises include, shorting the development life cycle of integrating or building a composed system, reducing the amount of effort needed for component selection process, and potentially contributing to the quality of the resultant system.

This paper explored potential problems and possible theories and presented a generic approach and model to predict reliability of a composed system. It highlighted differences between different models, and the importance of such models to COTS and component-based systems. Although, the paper focused on reliability predicting models, it also presented a generic approach to predicting non-functional properties of composed systems, which the author believes that it can be used to derive theories for predicting other non-functional properties, the most obvious of which is performance, for example.

The paper concludes that while many researchers have proposed many predication models and a significant progress has been made in this direction, yet, there are several challenges
need to be overcome before predication theories becomes widely used and adopted by industry.

7. References

<table>
<thead>
<tr>
<th>Reference</th>
<th>Title</th>
</tr>
</thead>
</table>